



*National Information Exchange Model*

# **Practical Implementer's Course**



United States  
Department of Justice

## **Technical Overview I**



# Practical Implementer's Course

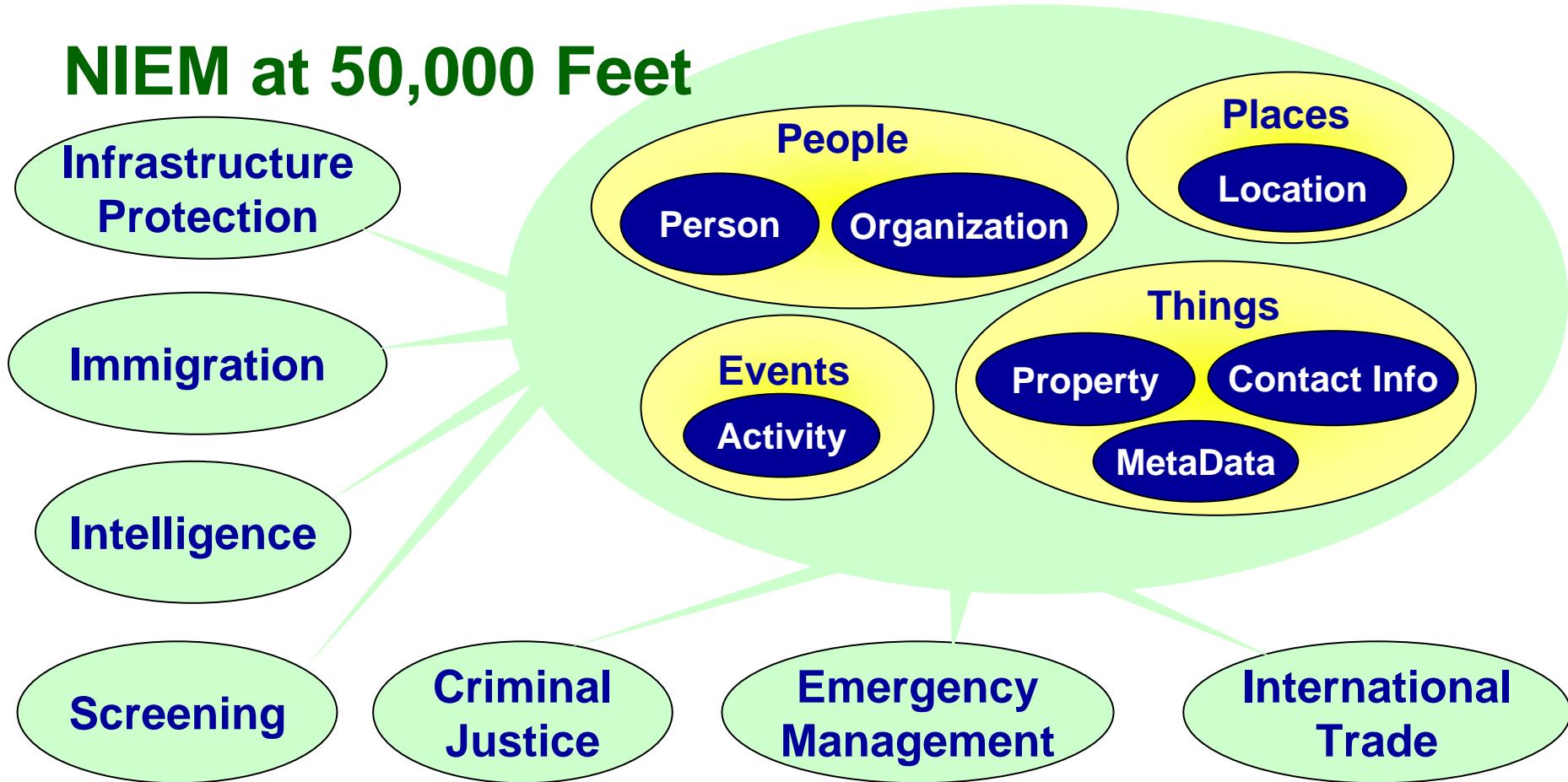
## Overview

- Technical architecture
- Namespaces
- Basic structure
- Component-naming rules
- Code tables
- Substitution Groups for Multiple Representations
- Extension Approaches



# Practical Implementer's Course

## NIEM at 50,000 Feet



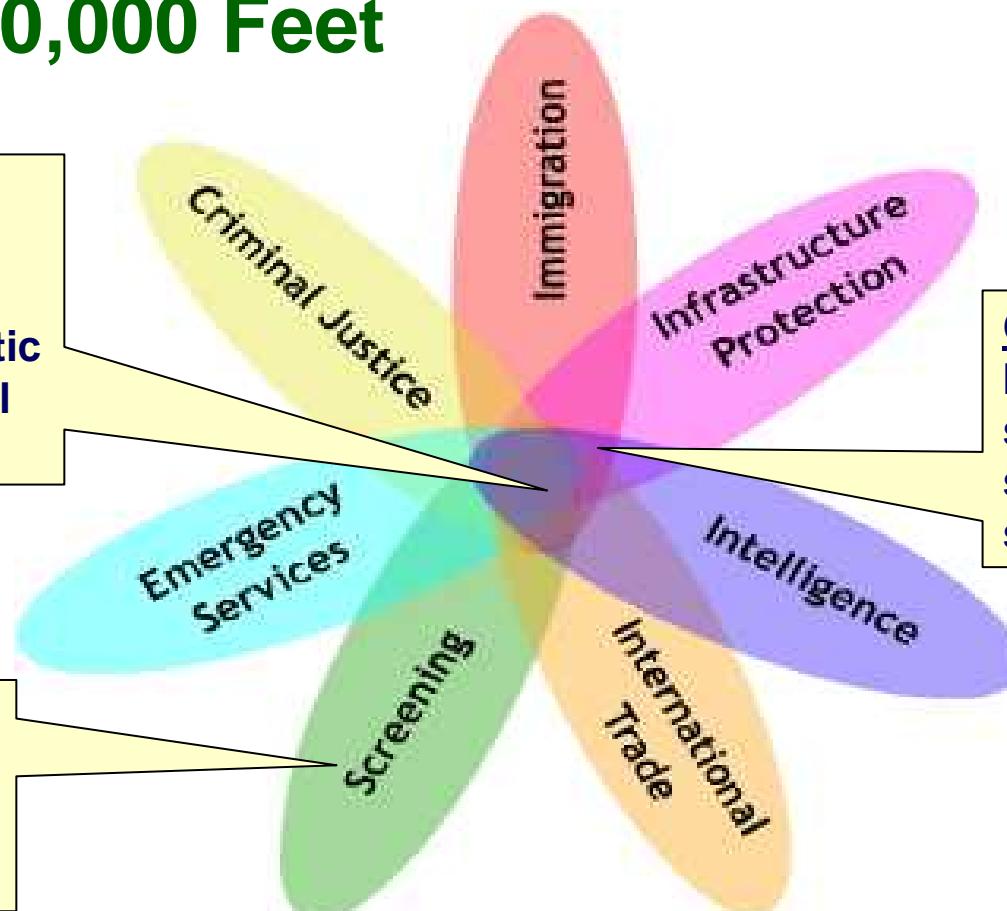


# Practical Implementer's Course

## NIEM At 20,000 Feet

**Universal & Structures:**

Items that exist & have same semantic meaning across all domains



**Domain Specific:**

Items with semantic meaning only in its domain

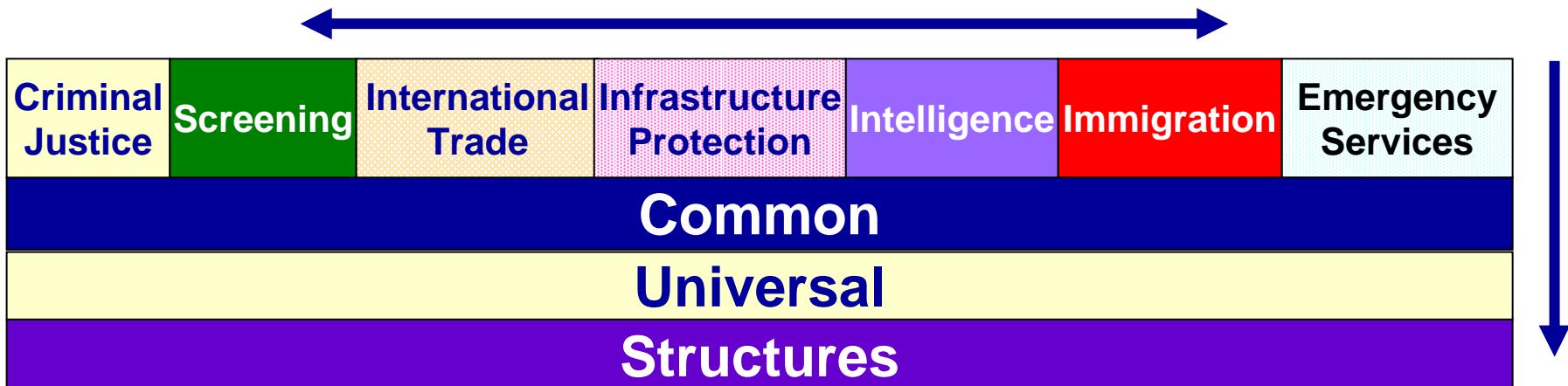
**Common:**  
Items with same semantic meaning shared across several domains



# Practical Implementer's Course

## NIEM Architecture

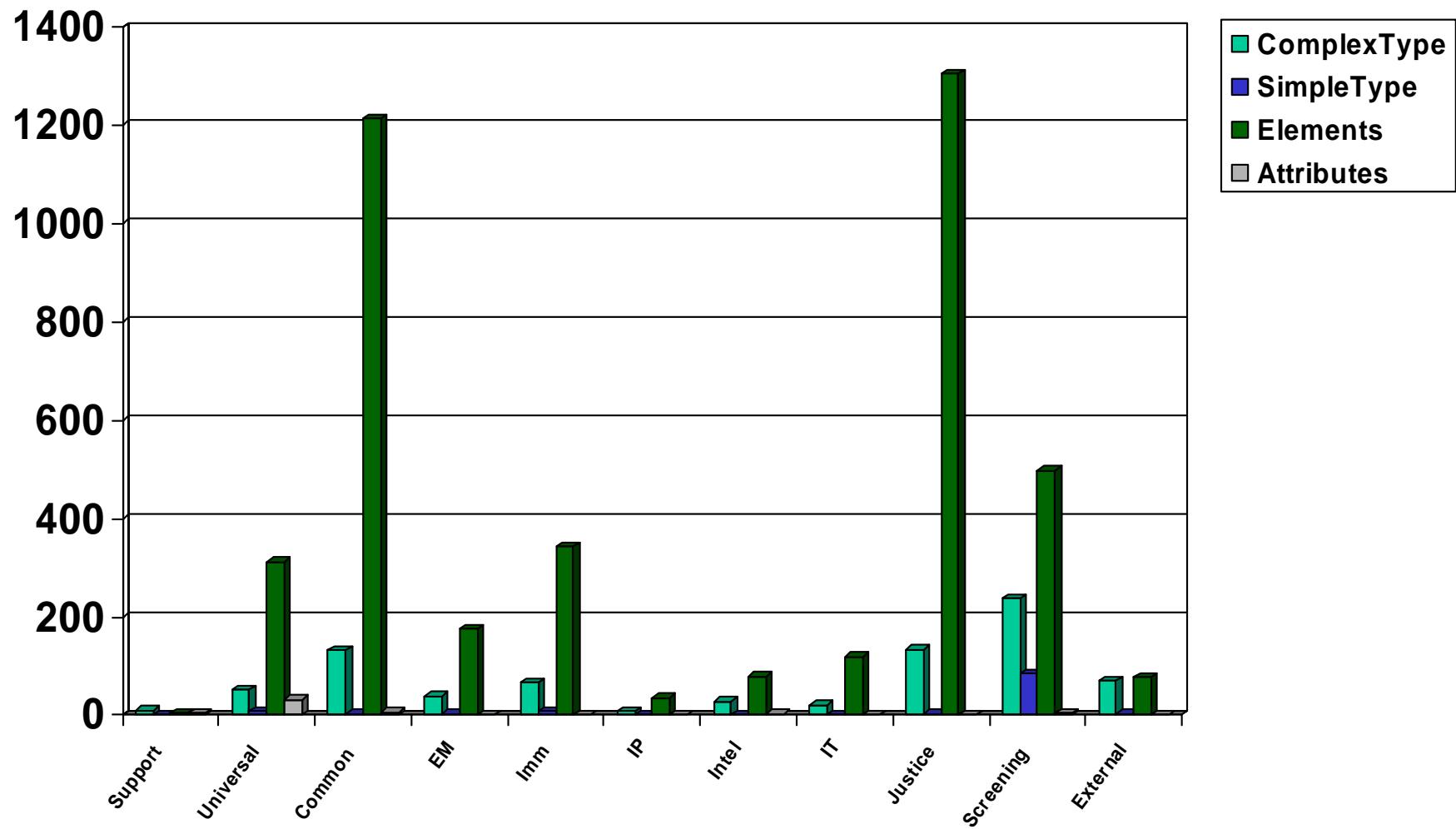
- NIEM is implemented as a layered architecture
  - Pieces are grouped in order of relevance
  - Follows a hierarchical usage order
    - Can only use items from same layer or lower layer





# Practical Implementer's Course

## NIEM Statistics





# Practical Implementer's Course

## Namespaces Revisited

- A convention to uniquely identify an item in a distributed network/architecture
- Namespaces are a base XML concept that are critical when working with NIEM
  - All layers and logical groupings are implemented in their own namespace



# Practical Implementer's Course

## Namespaces Revisited

**en · gi · neer** *[en-juh-neer]*  
—noun

The equivalent to  
an XML element

- 1.a person trained and skilled in the design, construction, and use of engines or machines, or in any of various branches of engineering: *a mechanical engineer; a civil engineer.*
- 2.a person who operates or is in charge of an engine.
- 3.*Also called locomotive engineer.* Railroads. a person who operates or is in charge of a locomotive.
- 4.a member of an army, navy, or air force specially trained in engineering work.

Designator indicating  
a different context

The specific semantic meaning of  
a word in a designated context



# Practical Implementer's Course

## Defining Namespaces

The definition of the local namespace for this document

```
<xsd:schema  
    targetNamespace="http://xml.org.gov/niem/1.0/extension"  
    xmlns:pmp="http://xml.org.gov/niem/1.0/extension"  
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
    xmlns:j="http://niem.gov/niem/domains/justice/1.0"  
    xmlns:niem-xsd="http://niem.gov/niem/proxy/xsd/1.0"  
    xmlns:c="http://niem.gov/niem/common/1.0"  
    xmlns:u="http://niem.gov/niem/universal/1.0"  
    xmlns:s="http://niem.gov/niem/structures/1.0" >
```

Other namespaces used in this document

Shortcut (or “Prefix”) to identify elements from a namespace

The full and unique designation for a namespace. It is called a URI (“Uniform Resource Identifier”)

NIEM  
Namespaces



# Practical Implementer's Course

## NIEM Namespace URI Naming

Host identifier which specifies the authoritative location for NIEM

Sub-location where NIEM is kept at the host

**xmlns:c="http://niem.gov/niem/common/1.0"**

The layer in the NIEM architecture

Version Number



# Practical Implementer's Course

## Using Namespaces: XSD

```
<xsd:extension base="j:PersonType">
  <xsd:sequence>
    <xsd:element ref="pmp:PharmacistNPINumber" minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="pmp:PharmacistStateLicense" minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="pmp:Residence" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:extension>
<xsd:element name="PharmacistNPINumber" type="u:IDType" nillable="true">
  <xsd:annotation>
    <xsd:documentation>
      Identifier assigned by the Centers for Medicare and Medi- Services (CMS).
    </xsd:documentation>
  </xsd:annotation>
</xsd:element>
```

“xsd” refers to W3C definition for schema document.  
defines what extension, element, sequence, etc mean.

“j” refers to justice layer

“pmp” refers to local namespace

“u” refers to the universal layer



# Practical Implementer's Course

## Using Namespaces: XML Instance

Namespaces are declared in an element which defines a scope

```
<SomeElement >
    xmlns:pmp="http://xml.org.gov/niem/1.0/extension"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:j="http://niem.gov/niem/domains/justice/1.0" >
    <pmp:Pharmacist>
        <u:PersonName>
            <u:PersonGivenName>John</u:PersonGivenName>
            <u:PersonSurName>Pillmaker</u:PersonSurName>
        </u:PersonName>
        <pmp:PharmacistsNPI>12345</pmp:PharmacistsNPI>
    </pmp:Pharmacist>
</SomeElement>
```

“pmp” refers to local namespace.

“u” refers to the universal layer.



# Practical Implementer's Course

## Practical Exercise - Namespaces

- Using the NIEM spreadsheet, how many namespaces define PersonType?
- Which namespaces are they?



# Practical Implementer's Course

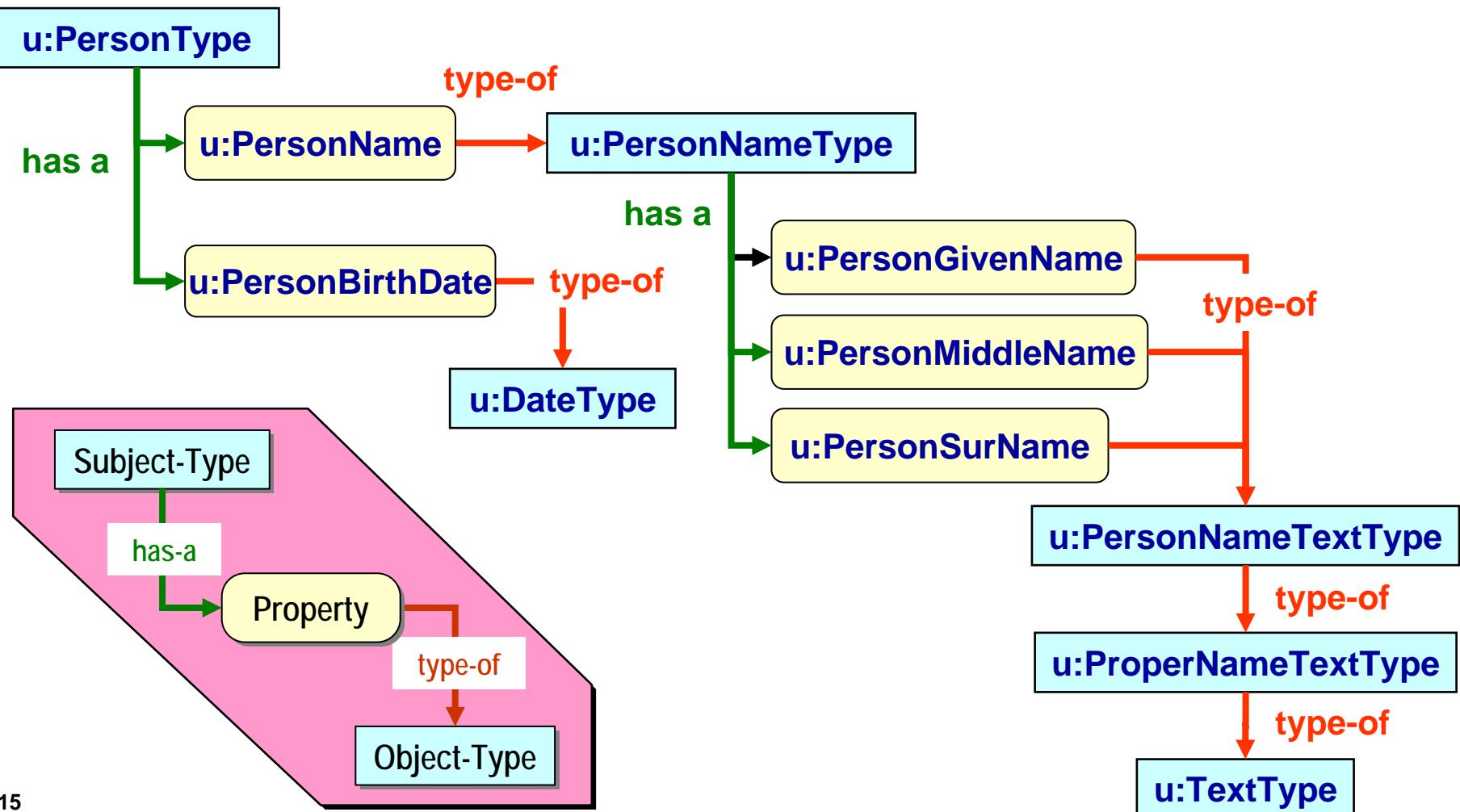
## Practical Exercise – Namespace Solution

- 7 definitions of PersonType exist in:
  - Universal
  - Common
  - Justice
  - Intelligence
  - International Trade
  - Immigration
  - Screening



# Practical Implementer's Course

## Conceptual Data Model—Structure





# Practical Implementer's Course

## How It Looks in XML

- **XML Elements**—define data semantics
- **XML Types**—define data structure

```
<PersonName>
```

Instance

```
    <PersonGivenName>John</PersonGivenName>
```

```
    <PersonSurName>Wandelt</PersonSurName>
```

```
</PersonName>
```

```
<xsd:complexType name="PersonNameType">
```

Schema

```
    <xsd:sequence>
```

```
        <element name="PersonGivenName" type="u:PersonNameTextType"/>
```

```
        <element name="PersonSurName" type="u:PersonNameTextType"/>
```

```
    </xsd:sequence>
```

```
</xsd:complexType>
```

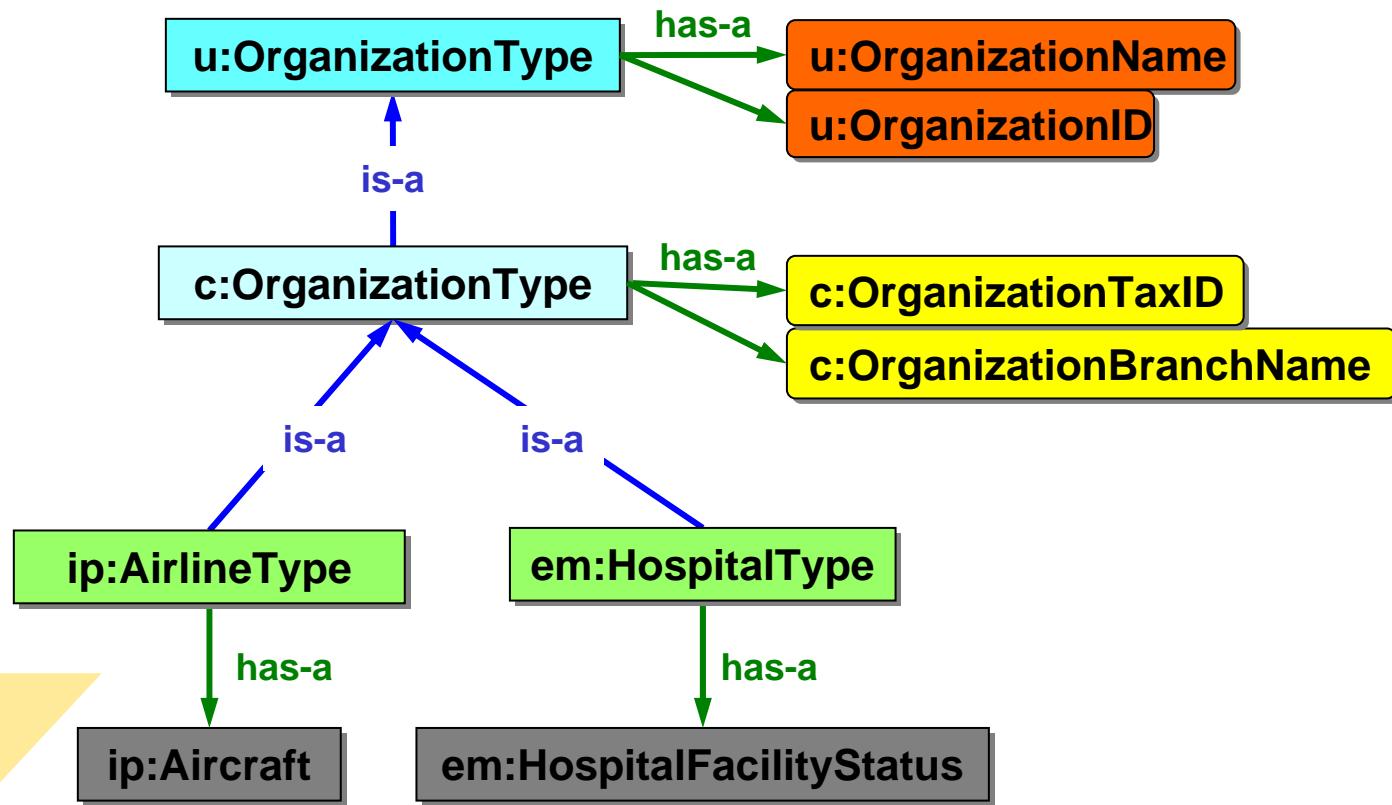
```
<xsd:element name="PersonName" type="u:PersonNameType"/>
```



# Practical Implementer's Course

## NIEM Object Model—Inheritance

General

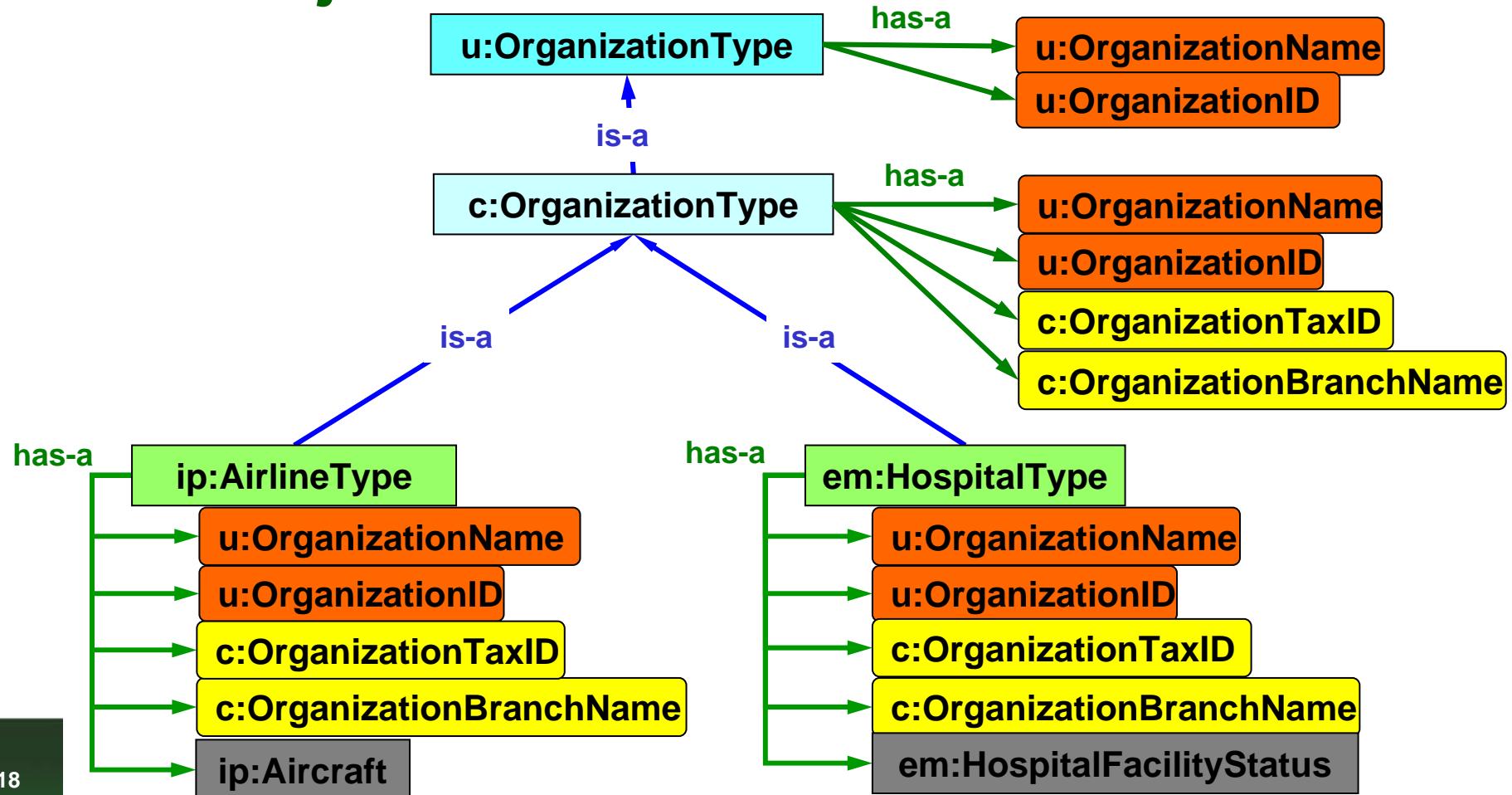


Specific



# Practical Implementer's Course

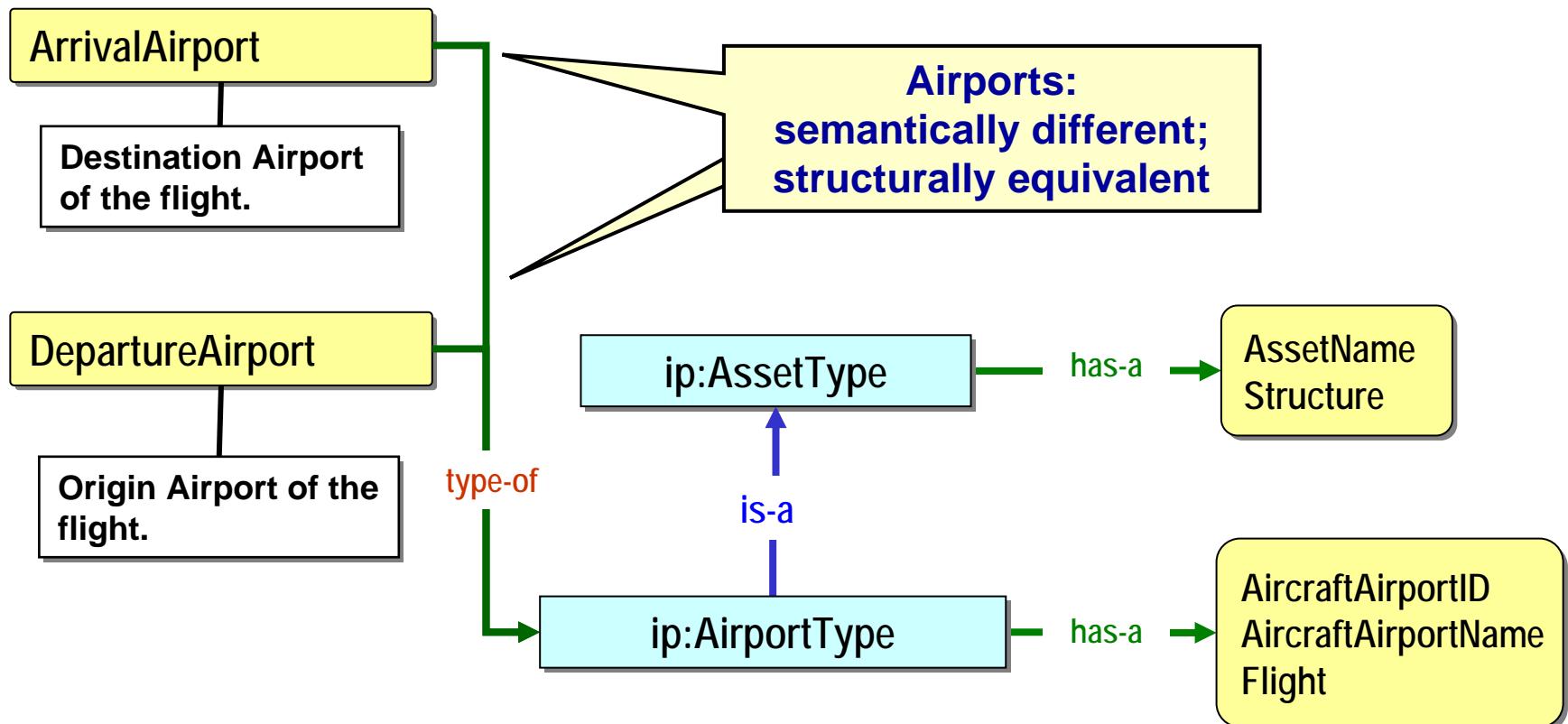
## NIEM Object Model—With Inheritance





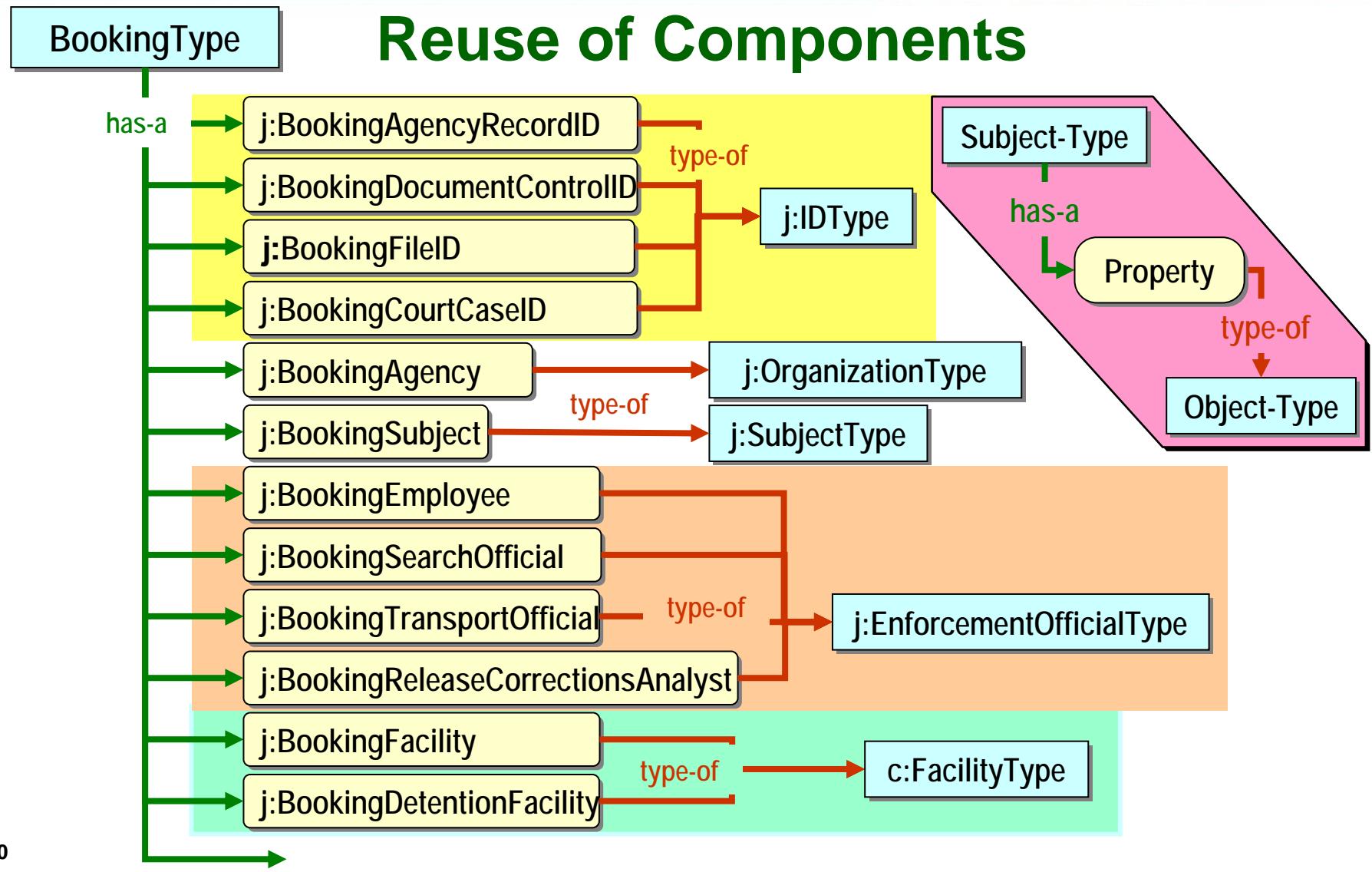
# Practical Implementer's Course

## Structural Equivalence/Semantic Difference





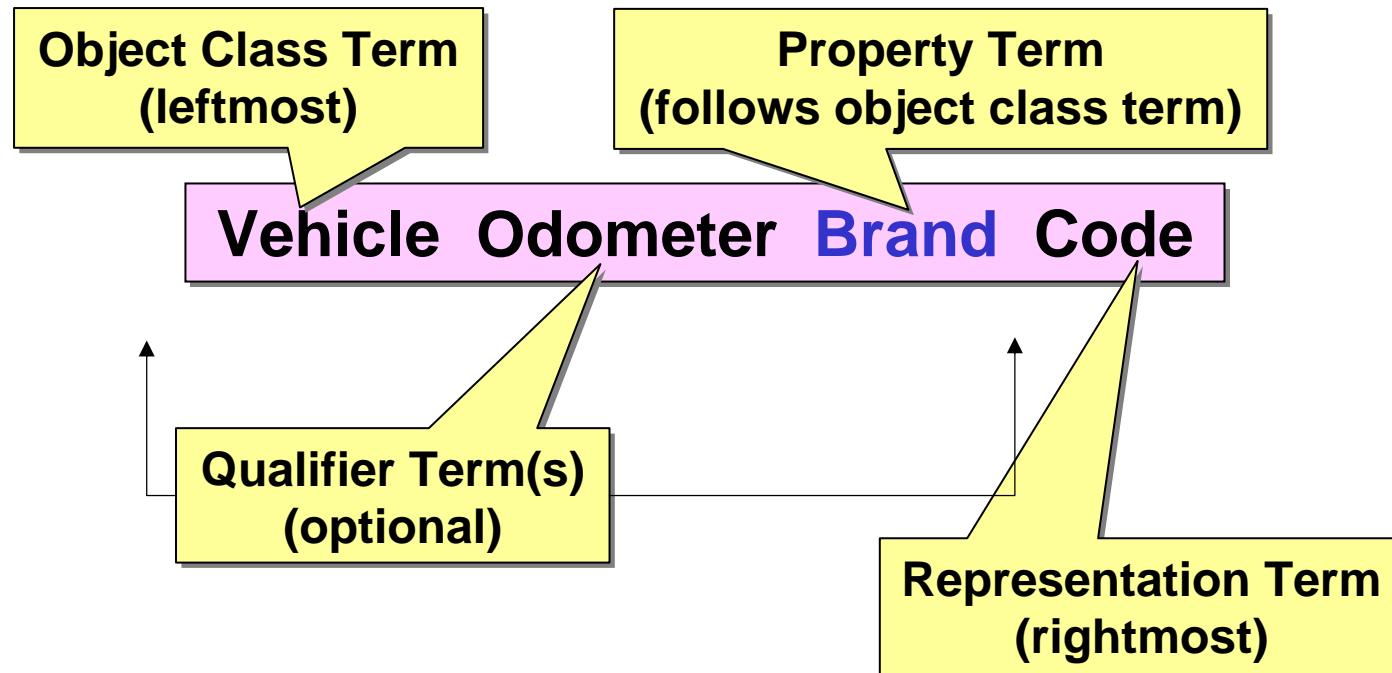
# Practical Implementer's Course





# Practical Implementer's Course

## Component Naming Rules





# Practical Implementer's Course

## Naming Rules Representation Terms

- Representation terms are from ebXML Core Component Tech Spec
- Includes
  - Amount
  - Binary Object
  - Code
  - DateTime
  - Identifier
  - Indicator
  - Measure
  - Numeric
  - Quantity
  - Text



# Practical Implementer's Course

## NIEM Context Definitions

### Schema Instance

```
<Arrest>
  ...
  <ActivityDate>2003-12-08</ActivityDate>
  ...
</Arrest>
```

- Data dictionary
  - ActivityDate—date of an activity that occurs at a singular point in time or a start date of an activity that occurs over a period of time
  - ArrestDate—date that a person is arrested, represented by ActivityDate in the context of Arrest property
  - BookingDate—date that a person is booked, represented by ActivityDate in the context of Booking property

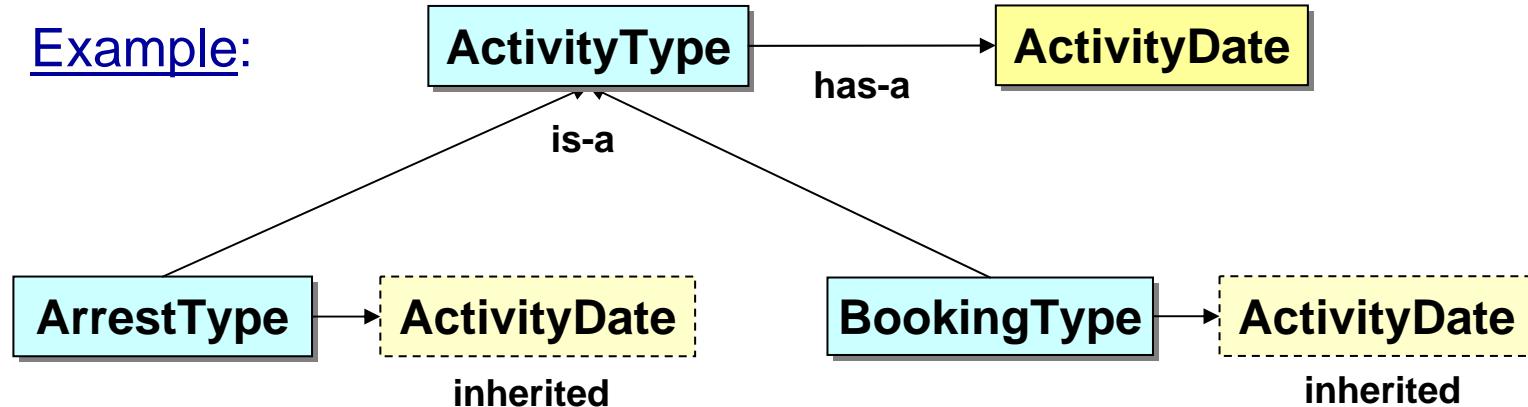


# Practical Implementer's Course

## NIEM Context Definitions (continued)

- Data Dictionary entries and their definitions that are not explicitly generated into the XML Schema specification for NIEM
- Alternate common names for properties that are inherited from a generic type to a more specific type (context)

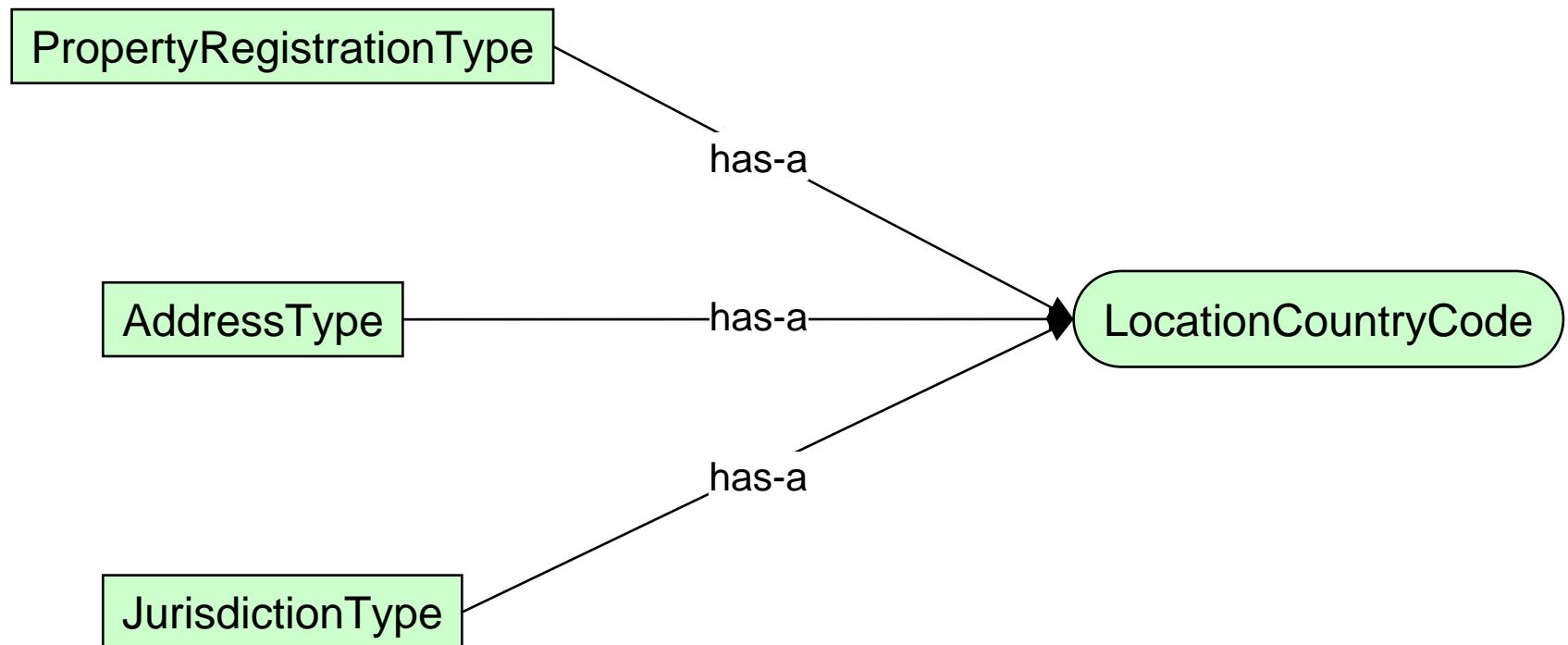
Example:





# Practical Implementer's Course

## An Element Can Have Multiple Subject Types





# Practical Implementer's Course

## Code Tables

- Many code tables are used within the community
- Some are extremely large (e.g., NCIC)
- Some are dynamic (e.g., vehicle model)
- Most fall under an administrative domain outside of NIEM governance
- All need to extend or supplement locally
- Often do not exist in an easily usable format



# Practical Implementer's Course

## External Code Tables

- NIEM uses separate namespaces for code tables
  - Facilitates code table ownership—when ready
  - Allows multiple tables for same conceptual property
  - Makes it easier to extend and add codes
  - Breaks up NIEM namespace into smaller parts
  - Allows validation of all codes against respective source table
  - Provides option for literal representation (TextType)



# Practical Implementer's Course

## External Code Tables (continued)

...

### Schema

```
<xsd:element name="LocationStateName" type="TextType" />
<xsd:element name="LocationStateUSPostalServiceCode"
    type="usps:USStateCodeType" />
```

...

...

### Instance

```
<u:LocationStateName>
    Alabama
</u:LocationStateName>
<c:LocationStateUSPostalServiceCode>
    AL
</c:LocationStateUSPostalServiceCode>
```

...

### Note:

FIPS 5-2 contains codes for areas around the U.S. that USPS does not



# Practical Implementer's Course

## Code List Exercise

- How many namespaces define a code for the state of Minnesota?
- What are the types in each of those namespaces?
  - Hint: Search for Facets in the NIEM SSGT

0 : 00

[www.niem.gov](http://www.niem.gov)



# Practical Implementer's Course

## Solution To Code List Exercise

- 9 code tables defined in 5 namespaces
  - ansi\_d20
    - JurisdictionAuthorityCodeSimpleType
  - fips\_10-4
    - InternationalStateCodeSimpleType
  - fips\_5.2
    - USStateCodeSimpleType
  - ncic
    - LISCodeSimpleType
    - LSTACodeSimpleType
    - POBCodeSimpleType
    - RESCodeSimpleType
  - usps\_states
    - USStateCodeSimpleType
    - USStateNumericCodeSimpleType

However...



# Practical Implementer's Course

## Solution Addendum

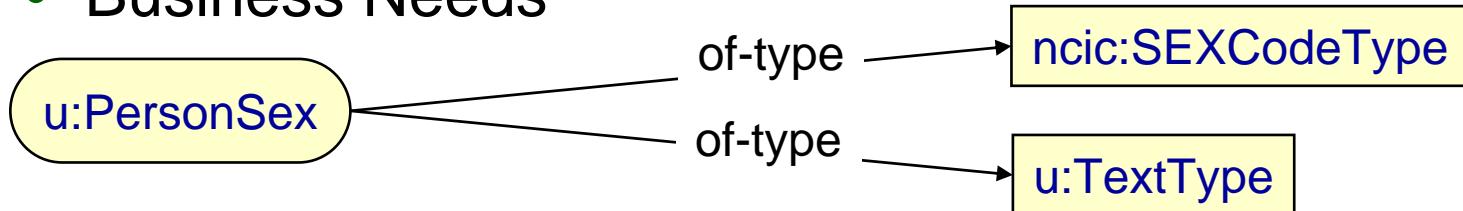
- 1 code table (StateNameType) defined in
  - <http://niem.gov/niem/external/urisa-street-address/draft-0.2.0/dhs-gmo/1.0.0>
    - In file street\_address\_data\_standard.xsd
- Remember that tools may have unexpected limitations so don't necessarily rely on a single search tool or methodology



# Practical Implementer's Course

## Multiple Representations Problem

- The same element may be represented by different types depending on
  - Domain
  - Business Needs



**NIEM leverages the XML Concept of “Substitution Groups” to address this problem.**



# Practical Implementer's Course

## Substitution Groups

- Allow elements to be substituted for other elements
- Elements are assigned to a special group of elements that can be substituted for a particular named element called the head element
  - Head element must be declared as global
- Elements must be
  - Same type as the head element, or
  - Of type derived from head element's type



# Practical Implementer's Course

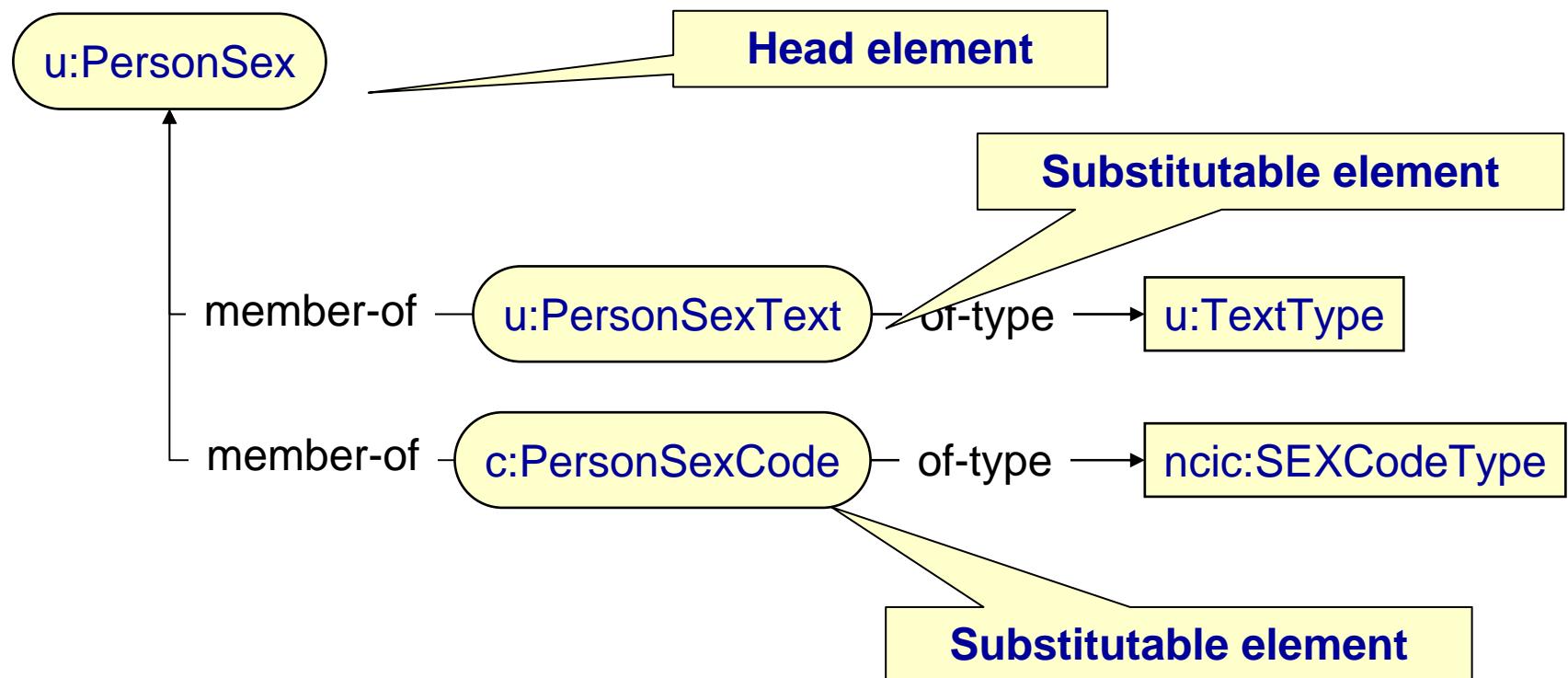
## Abstract Elements

- Elements declared as abstract cannot be used in an XML instance
- Forces substitution
  - A member of element's substitution group **must** appear in the instance.
- In NIEM these serve as the head element of substitution groups



# Practical Implementer's Course

## NIEM Structure for PersonSex





# Practical Implementer's Course

## NIEM Schema for PersonSex

universal

```
<xsd:complexType name="PersonType">
  <xsd:complexContent>
    <xsd:extension base="u:SuperType">
      <xsd:sequence>
        <xsd:element ref="u:PersonSex" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

common

```
<xsd:element name="Person" type="u:PersonType">
<xsd:element name="PersonSex" abstract="true"/>
<xsd:element name="PersonSexText" type="u:TextType"
  substitutionGroup="u:PersonSex" />
<xsd:element name="PersonSexCode" type="u:SEXCodeType"
  substitutionGroup="u:PersonSex" />
```

Head element declared globally and abstract

Substitution group membership



# Practical Implementer's Course

## Valid XML Instances Using PersonSex

```
<u:Person>
    <c:PersonSexCode>F</c:PersonSexCode>
</u:Person>
```

Substituted for the  
abstract element  
u:PersonSex

```
<u:Person>
    <u:PersonSexText>Female</u:PersonSexText>
</u:Person>
```



# Practical Implementer's Course

## Substitution Groups Practical Exercise 1

- Using the NIEM spreadsheet, what is the head element for the substitution group designating a country?
- What are all of the members (substitutable elements) of the substitution group and what are their types?



# Practical Implementer's Course

## Substitution Groups Solution 1

- Head Element: **u:LocationCountry**

### Substitutable Element

c:LocationCountryFIPS10-4Code

intel:LocationCountryFIPS10-4PlusNCTCCode

scr:LocationCountryID

c:LocationCountryISO3166Alpha2Code

c:LocationCountryISO3166Alpha3Code

c:LocationCountryISO3166NumericCode

u:LocationCountryName

### Type

fips\_10-4:CountryCodeType

u:TextType

u:IDType

iso\_3166:CountryAlpha2CodeType

iso\_3166:CountryAlpha3CodeType

iso\_3166:CountryNumericCodeType

u:ProperNameTextType



# Practical Implementer's Course

## Substitution Groups Practical Exercise 2

- Given the schema below, come up with 2 instance documents representing PersonSecurityClearance in 2 different ways
  - Specify a person name of your own choice

```
<element name="PersonSecurityClearance" abstract="true"/>
<element name="PersonSecurityClearanceText" type="string" substitutionGroup="PersonSecurityClearance"/>
<element name="PersonSecurityClearanceCode" type="SecurityClassificationLevelType"
       substitutionGroup="PersonSecurityClearance"/>
<simpleType name="SecurityClassificationLevelType">
    <restriction base="token">
        <enumeration value="Unclassified"/>
        <enumeration value="Restricted"/>
        <enumeration value="Confidential"/>
        <enumeration value="Secret" />
        <enumeration value="Would Have To Kill You If I Told You"/>
    </restriction>
</simpleType>
<element name="Person" type="PersonType"/>
<element name="PersonName" type="string"/>
<complexType name="PersonType">
    <sequence>
        <element ref="PersonName" minOccurs="0" maxOccurs="unbounded"/>
        <element ref="PersonSecurityClearance" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
</complexType>
```

0 : 00



# Practical Implementer's Course

## Substitution Groups Solution 2

```
<Person>
    <PersonName>Tom Carlson</PersonName>
    <PersonSecurityClearanceText>
        None
    </PersonSecurityClearanceText>
</Person>

<Person>
    <PersonName>Joe Mierwa</PersonName>
    <PersonSecurityClearanceCode>
        Secret
    </PersonSecurityClearanceCode>
</Person>
```



# Practical Implementer's Course

## Key Substitution Groups Usage

- Code Tables
- Measurements & Measurement ranges
  - Age
  - Speed
  - etc
- Augmentation
- External Standards Wrappers



# Practical Implementer's Course

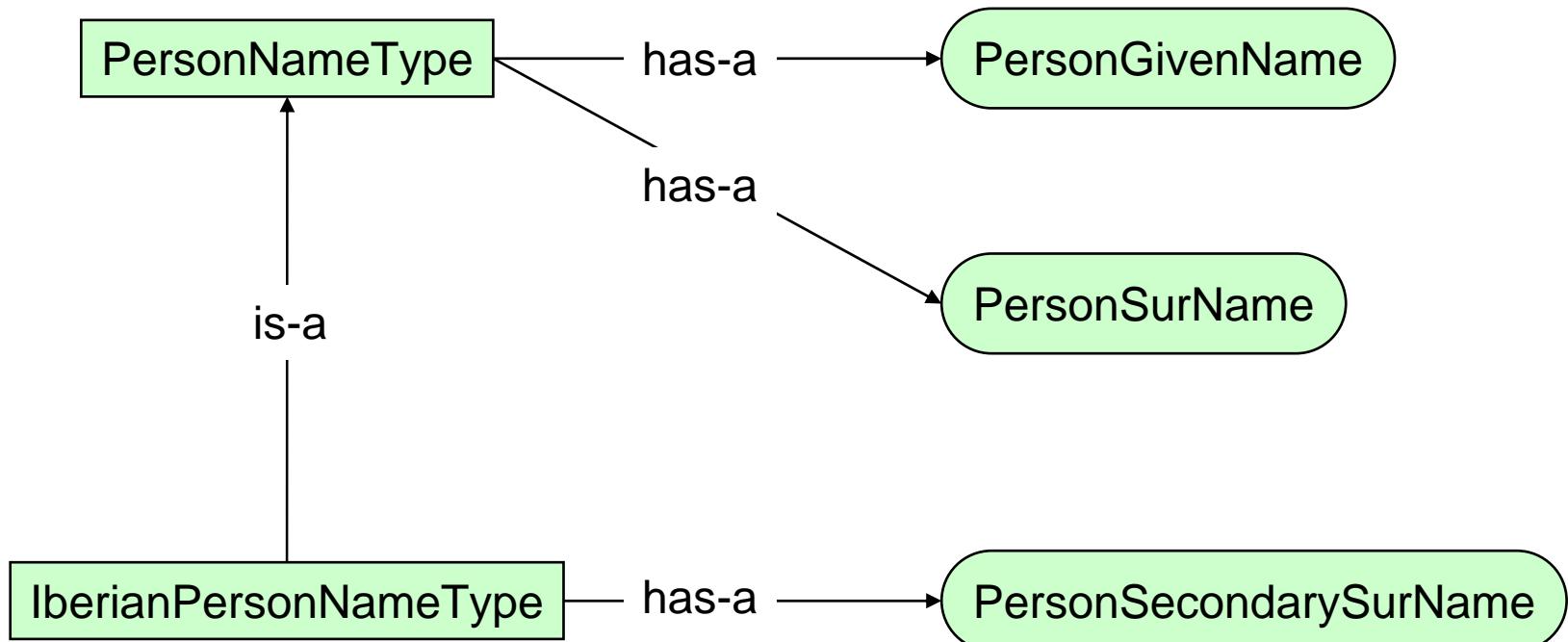
## Why Extension and Substitution?

- No single group can possibly think of everything
  - Users have additional needs
  - Different information requirements
  - Extensions can be made with NIEM
- Allows partial understanding
  - An application that understands PersonType can understand much of any type derived from PersonType
- Basic functionality from XML Schema
  - We did not invent the extension and substitution mechanism
  - We just use what is already standard



# Practical Implementer's Course

## Type Extension





# Practical Implementer's Course

## Dynamic Type Substitution

- Dynamic Type Substitution involves creating a new type in XML Schema, based on existing NIEM types. Then, **in an instance document**, an element is reassigned to that new type via the xsi:type construct to substitute the new type for the usual type
- For example, you could create an IberianPersonNameType to hold the usual PersonName information plus a PersonSecondarySurName. In your instance document, you would use a PersonName object but declare its type to be IberianPersonNameType instead



# Practical Implementer's Course

## Dynamic Type Substitution—Schema

```
<xs:complexType name="IberianPersonNameType">
  <xs:complexContent>
    <xs:extension base="u:PersonNameType">
      <xs:sequence>
        <xs:element name="PersonSecondarySurName" type="u:PersonNameTextType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

This type gets specified in the instance document



# Practical Implementer's Course

## Dynamic Type Substitution—Instance Document

Indicates the type that PersonName really is

```
<u:PersonName xsi:type="tns:IberianPersonNameType">  
  <u:PersonGivenName>Gabriel</u:PersonGivenName>  
  <u:PersonSurName>Garcia</u:PersonSurName>  
  <tns:PersonSecondarySurName>Marquez</tns:PersonSecondarySurName>  
</u:PersonName>
```

Can use this element because  
xsi:type was specified above



# Practical Implementer's Course

## Dynamic Type Substitution Practical Exercise

- Given the schema snippet for the “tns” namespace below, come up with an instance document that includes a secondary surname:

```
<complexType name="PersonNameType">
  <complexContent>
    <extension base="u:PersonNameType">
      <sequence>
        <element name="PersonSecondarySurName" type="u:PersonNameTextType" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
```



# Practical Implementer's Course

## Dynamic Type Substitution Solution

```
<u:Person>
  <u:PersonName xsi:type="tns:PersonNameType">
    <u:PersonGivenName>Marge</u:PersonGivenName>
    <u:PersonSurName>Simpson</u:PersonSurName>
    <tns:PersonSecondarySurName>
      Bouvier
    </tns:PersonSecondarySurName>
  </u:PersonName>
</u:Person>
```



# Practical Implementer's Course

## Concrete Extension

- Concrete Extension involves creating a new type by extending an existing NIEM type in XML Schema to include additional local elements. Then, in an instance document, elements of this new type are now in a local namespace, **along with higher level elements** that contain them
- For example, you could create an IberianPersonNameType to hold the usual PersonName information plus a PersonSecondarySurName. In your instance document, a PersonSecondarySurName would be in your local namespace. Additionally, a PersonName and Person will also be in your local namespace



# Practical Implementer's Course

## Concrete Extension—Schema

```
<xs:complexType name="IberianPersonNameType">
  <xs:complexContent>
    <xs:extension base="u:PersonNameType">
      <xs:sequence>
        <xs:element name="PersonSecondarySurName" type="u:PersonNameTextType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="IberianPerson">
  <xs:complexContent>
    <xs:extension base="u:PersonType">
      <xs:sequence>
        <xs:element name="PersonIberianName" type="tns:IberianPersonNameType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Need to extend person as well  
so the explicit type can be  
referenced in the XML instance



# Practical Implementer's Course

## Concrete Extension—Instance Document

Instance explicitly references  
the extended type

```
<tns:IberianPerson>
  <tns:PersonIberianName>
    <u:PersonGivenName>Gabriel</u:PersonGivenName>
    <u:PersonSurName>Garcia</u:PersonSurName>
    <tns:PersonSecondarySurName>Marquez</tns:PersonSecondarySurName>
  </tns:PersonIberianName>
</tns:IberianPerson>
```



# Practical Implementer's Course

## Extension Versus Substitution

- Concrete extension
  - Works easily in Web services environments
  - Schema shows exactly what you will get
  - More elements in the local namespace, so...
    - More work to create
    - More elements being redefined, some of which are not changed
- Dynamic type substitution
  - Less elements in local namespace
  - Happens in the instance document so can't see where the substitution occurs at schema level
  - Has implications for Web services use
  - Hard for XSD validation
  - Need to document substitution outside of schema



# Practical Implementer's Course

## Concrete Type Extension Practical Exercise

- Given the schema snippet for the “tns” namespace below, come up with an instance document that includes a secondary surname:

```
<complexType name="PersonNameType">
  <complexContent>
    <extension base="u:PersonNameType">
      <sequence>
        <element name="PersonSecondarySurName" type="u:PersonNameTextType" />
      </sequence>
    </extension>
  </complexContent>
</complexType>

<element name="Person">
  <complexType>
    <complexContent>
      <extension base="u:PersonType">
        <sequence>
          <element name="PersonName" type="tns:PersonNameType" />
        </sequence>
      </extension>
    </complexContent>
  </complexType>
</element>
```

0 : 00



# Practical Implementer's Course

## Concrete Type Extension Solution

```
<tns:Person>
  <tns:PersonName>
    <u:PersonGivenName>Marge</u:PersonGivenName>
    <u:PersonSurName>Simpson</u:PersonSurName>
    <tns:PersonSecondarySurName>
      Bouvier
    </tns:PersonSecondarySurName>
  </tns:PersonName>
</tns:Person>
```



# Practical Implementer's Course



This work is licensed under the Creative Commons Attribution-ShareAlike 2.5 License.

To view a copy of this license

- a) visit <http://creativecommons.org/licenses/by-sa/2.5/>; or,
- b) send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA."